ATTORNEY'S DOCKET NO. 06502.0365
CLIENT DOCKET NO. P5188
PATENTS
EXPRESS MAIL NO. EL396216914US

UNITED STATES PATENT APPLICATION

OF

GUY L. STEELE JR.

FOR

FLOATING POINT ADDER WITH EMBEDDED STATUS INFORMATION

RELATED APPLICATIONS

[001] U.S. Patent Application Serial No. _____, filed on even date herewith in the name of Guy L. Steele Jr. and entitled "Floating Point System That Represents Status Flag Information Within a Floating Point Operand," assigned to the assignee of the present application, is hereby incorporated by reference.

DESCRIPTION OF THE INVENTION

Field of the Invention

[002] The invention relates generally to systems and methods for performing floating point operations, and more particularly to systems and methods for performing floating point addition with embedded status information associated with a floating point operand.

Background of the Invention

[003] Digital electronic devices, such as digital computers, calculators and other devices, perform arithmetic calculations on values in integer, or "fixed point," format, in fractional, or "floating point" format, or both. Institute of Electrical and Electronic Engineers (IEEE) Standard 754, (hereinafter "IEEE Std. 754" or "the Standard") published in 1985 and adopted by the American National Standards Institute (ANSI), defines several standard formats for expressing values in floating point format and a number of aspects regarding behavior of computation in connection therewith. In accordance with IEEE Std. 754, a representation in floating point format comprises a plurality of binary digits, or "bits," having the structure

[004] $$se_{msb}\cdots e_{lsb}f_{msb}\cdots f_{lsb}$$

where bit "s" is a sign bit indicating whether the entire value is positive or negative, bits "$e_{msb} \cdots e_{lsb}$" comprise an exponent field represent the exponent "e" in unsigned binary biased format, and bits "$f_{msb} \cdots f_{lsb}$" comprise a fraction field that represents the fractional portion "f" in unsigned binary format ("msb" represents "most significant bit" and "lsb" represents "least significant bit"). The Standard defines two general formats. A "single" format comprises thirty-two bits while a "double" format comprises sixty-four bits. In the single format, there is one sign bit "s," eight bits "$e_7 \ldots e_0$" comprising the exponent field and twenty-three bits "$f_{22} \ldots f_0$" comprising the fraction field. In the double format, there is one sign bit "s," eleven bits "$e_{10} \ldots e_0$" comprising the exponent field and fifty-two bits "$f_{51} \ldots f_0$" comprising the fraction field.

[005] As indicated above, the exponent field of the floating point representation "$e_{msb} \cdots e_{lsb}$" represents the exponent "E" in biased format. The biased format provides a mechanism by which the sign of the exponent is implicitly indicated. In particular, the bits "$e_{msb} \cdots e_{lsb}$" represent a binary encoded value "e" such that "e=E + bias." This allows the exponent E to extend from -126 to +127, in the eight-bit "single" format, and from -1022 to +1023 in the eleven-bit "double" format, and provides for relatively easy manipulation of the exponents in multiplication and division operations, in which the exponents are added and subtracted, respectively.

[006] IEEE Std. 754 provides for several different formats with both the single and double formats which are generally based on the bit patterns of the bits "$e_{msb} \cdots e_{lsb}$" comprising the exponent field and the bits "$f_{msb} \cdots f_{lsb}$" comprising the fraction field. If a number is represented such that all of the bits "$e_{msb} \cdots e_{lsb}$" of the

exponent field are binary one's (*i.e.*, if the bits represent a binary-encoded value of "255" in the single format or "2047" in the double format) and all of the bits "$f_{msb} \cdots f_{lsb}$" of the fraction field are binary zeros, then the value of the number is positive or negative infinity, depending on the value of the sign bit "s." In particular, the value "v" is $v = (-1)^s \infty$, where "∞" represents the value "infinity." On the other hand, if all of the bits "$e_{msb} \cdots e_{lsb}$" of the exponent field are binary one's and if the bits "$f_{msb} \cdots f_{lsb}$" of the fraction field are not all zero's, then the value that is represented is deemed "not a number," which is abbreviated in the Standard by "NaN."

[007] If a number has an exponent field in which the bits "$e_{msb} \cdots e_{lsb}$" are neither all binary ones nor all binary zeros (*i.e.,* if the bits represent a binary-encoded value between 1 and 254 in the single format or between 1 and 2046 in the double format), the number is said to be a "normalized" format. For a number in the normalized format, the value represented by the number is $v = (-1)^s \, 2^{\text{e-bias}} (1.\left| f_{msb} \cdots f \cdots_{lsb} \right.)$, where "|" represents a concatenation operation. Effectively, in the normalized format, there is an implicit most significant digit having the value "one," so that the twenty-three digits in the fraction field of the single format, or the fifty-two digits in the fraction field of the double format, will effectively represent a value having twenty-four digits or fifty-three digits of precision, respectively, where the value is less than two, but not less than one.

[008] On the other hand, if a number has an exponent field in which the bits "$e_{msb} \cdots e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are not all zero, the number is said to be a "de-normalized" format. For a number in the de-normalized format, the value

represented by the number is $v = (-1)^s 2^{e-bias+1}(0.|f_{msb} \cdots f_{lsb})$. It will be appreciated that the range of values of numbers that can be expressed in the de-normalized format is disjoint from the range of values of numbers that can be expressed in the normalized format, for both the single and double formats. Finally, if a number has an exponent field in which the bits "$e_{msb} \cdots e_{lsb}$" are all binary zeros, representing the binary-encoded value of "zero," and a fraction field in which the bits $f_{msb} \cdots f_{lsb}$ are all zero, the number has the value "zero". It will be appreciated that the value "zero" may be positive zero or negative zero, depending on the value of the sign bit.

[009] Generally, circuits or devices that perform floating point computations or operations (generally referred to as floating point units) conforming to IEEE Std. 754 are designed to generate a result in three steps:

[010] (a) In the first step, an approximation calculation step, an approximation to the absolutely accurate mathematical result (assuming that the input operands represent the specific mathematical values as described by IEEE Std. 754) is calculated that is sufficiently precise as to allow this accurate mathematical result to be summarized. The summarized result is usually represented by a sign bit, an exponent (typically represented using more bits than are used for an exponent in the standard floating-point format), and some number "N" of bits of the presumed result fraction, plus a guard bit and a sticky bit. The value of the exponent will be such that the value of the fraction generated in step (a) consists of a 1 before the binary point and a fraction after the binary point. The bits are commonly calculated so as to obtain the same result as the following conceptual procedure (which is impossible under some circumstances to carry out in practice):

calculate the mathematical result to an infinite number of bits of precision in binary scientific notation, and in such a way that there is no bit position in the significand such that all bits of lesser significance are 1-bits (this restriction avoids the ambiguity between, for example, 1.100000... and 1.011111... as representations of the value "one-and-one-half"); let the N most significant bits of the infinite significand be used as the intermediate result significand; let the next bit of the infinite significand be the guard bit; and let the sticky bit be 0 if and only if ALL remaining bits of the infinite significant are 0-bits (in other words, the sticky bit is the logical OR of all remaining bits of the infinite fraction after the guard bit).

[011] (b) In the second step, a rounding step, the guard bit, the sticky bit, perhaps the sign bit, and perhaps some of the bits of the presumed significand generated in step (a) are used to decide whether to alter the result of step (a). For conventional rounding modes defined by IEEE Std. 754, this is a decision as to whether to increase the magnitude of the number represented by the presumed exponent and fraction generated in step (a). Increasing the magnitude of the number is done by adding 1 to the significand in its least significant bit position, as if the significand were a binary integer. It will be appreciated that, if the significand is all 1-bits, then the magnitude of the number is "increased" by changing it to a high-order 1-bit followed by all 0-bits and adding 1 to the exponent.

[012] Regarding the rounding modes, it will be further appreciated that,

[013] (i) if the result is a positive number, and

[014] (a) if the decision is made to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up (*i.e.,* towards positive infinity), but

[015] (b) if the decision is made not to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down (*i.e.,* towards negative infinity); and

[016] (ii) if the result is a negative number, and

[017] (a) if the decision is made to increase, effectively the decision has been made to decrease the value of the result, thereby rounding the result down, but

[018] (b) if the decision is made not to increase, effectively the decision has been made to increase the value of the result, thereby rounding the result up.

[019] (c) In the third step, a packaging step, a result is packaged into a standard floating-point format. This may involve substituting a special representation, such as the representation defined for infinity or NaN if an exceptional situation (such as overflow, underflow, or an invalid operation) was detected. Alternatively, this may involve removing the leading 1-bit (if any) of the fraction, because such leading 1-bits are implicit in the standard format. As another alternative, this may involve shifting the fraction in order to construct a denormalized number. As a specific example, it is assumed that this is the step that forces the result to be a NaN if any input operand is a NaN. In this step, the decision is also made as to whether the result should be an infinity. It will be appreciated that, if the

result is to be a NaN or infinity from step (b), the original result will be discarded and an appropriate representation will be provided as the result.

[020] In addition in the packaging step, floating point status information is generated, which is stored in a floating point status register. The floating point status information generated for a particular floating point operation includes indications, for example, as to whether

[021] (i) a particular operand is invalid for the operation to be performed ("invalid operation");

[022] (ii) if the operation to be performed is division, the divisor is zero ("division-by-zero");

[023] (iii) an overflow occurred during the operation ("overflow");

[024] (iv) an underflow occurred during the operation ("underflow"); and

[025] (v) the rounded result of the operation is not exact ("inexact").

[026] These conditions are typically represented by flags that are stored in the floating point status register. The floating point status information can be used to dynamically control the operations in response to certain instructions, such as conditional branch, conditional move, and conditional trap instructions that may be in the instruction stream subsequent to the floating point instruction. Also, the floating point status information may enable processing of a trap sequence, which will interrupt the normal flow of program execution. In addition, the floating point status information may be used to affect certain ones of the functional unit control signals

that control the rounding mode. IEEE Std. 754 also provides for accumulating floating point status information from, for example, results generated for a series or plurality of floating point operations.

[027] IEEE Std. 754 has brought relative harmony and stability to floating-point computation and architectural design of floating-point units. Moreover, its design was based on some important principles, and rests on a sensible mathematical semantics that eases the job of programmers and numerical analysts. It also supports the implementation of interval arithmetic, which may prove to be preferable to simple scalar arithmetic for many tasks. Nevertheless, IEEE Std. 754 has some serious drawbacks, including:

[028] (i) Modes (*e.g.*, the rounding modes and traps enabled/disabled mode), flags (*e.g.*, flags representing the status information), and traps required to implement IEEE Std. 754 introduce implicit serialization issues. Implicit serialization is essentially the need for serial control of access (read/write) to and from globally used registers, such as a floating point status register. Under IEEE Std. 754, implicit serialization may arise between (1) different concurrent floating-point instructions and (2) between floating point instructions and the instructions that read and write the flags and modes. Furthermore, rounding modes may introduce implicit serialization because they are typically indicated as global state, although in some microprocessor architectures, the rounding mode is encoded as part of the instruction operation code, which will alleviate this problem to that extent. Thus, the potential for implicit serialization makes the Standard difficult to

implement coherently and efficiently in today's superscalar and parallel processing architectures without loss of performance.

[029] (ii) The implicit side effects of a procedure that can change the flags or modes can make it very difficult for compilers to perform optimizations on floating point code. As a result, compilers for most languages usually assume that every procedure call is an optimization barrier in order to be safe. This unfortunately may lead to further loss of performance.

[030] (iii) Global flags, such as those that signal certain modes, make it more difficult to do instruction scheduling where the best performance is provided by interleaving instructions of unrelated computations. Thus, instructions from regions of code governed by different flag settings or different flag detection requirements cannot easily be interleaved when they must share a single set of global flag bits.

[031] (iv) Furthermore, traps have been difficult to integrate efficiently into computing architectures and programming language designs for fine-grained control of algorithmic behavior.

[032] Thus, there is a need for a system that avoids such problems when performing floating point operations and, in particular, when performing floating point addition with embedded status information associated with a floating point operand.

SUMMARY OF THE INVENTION

[033] Consistent with the current invention, a floating point adder with embedded status information method and system are provided that avoid the

problems associated with prior art floating point adder systems as discussed herein above.

[034] In one aspect, a system for providing a floating point sum comprises an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. In addition, the system comprises a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the sum of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[035] In another aspect, a method for providing a floating point sum comprises determining a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. In addition, the method comprises asserting a resulting floating point operand containing the sum of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[036] In yet another aspect, a computer-readable medium on which is stored a set of instructions for providing a floating point sum, which when executed perform stages comprising determining a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand respectively. The

instruction set further includes asserting a resulting floating point operand containing the sum of the first floating point operand and the second floating point operand and a resulting status embedded within the resulting floating point operand.

[037] Both the foregoing general description and the following detailed description are exemplary and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

[038] The accompanying drawings provide a further understanding of the invention and, together with the detailed description, explain the principles of the invention. In the drawings:

[039] FIG. 1 is a functional block diagram of an exemplary system for providing a floating point sum consistent with an embodiment of the present invention;

[040] FIG. 2 illustrates exemplary formats for representations of floating point values generated by the system of FIG. 1 consistent with an embodiment of the present invention;

[041] FIG. 3 illustrates a table useful in understanding the operations of the exemplary system of FIG. 1 consistent with an embodiment of the present invention; and

[042] FIGs. 4A through 4E depict exemplary patterns of input and output signals received and generated by an addition decision table logic circuit used in the exemplary system of FIG. 1 consistent with an embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

[043] Reference will now be made to various embodiments according to this invention, examples of which are shown in the accompanying drawings and will be obvious from the description of the invention. In the drawings, the same reference numbers represent the same or similar elements in the different drawings whenever possible.

[044] Related U.S. Patent Application Serial No. _____, which has previously been incorporated by reference, describes an exemplary floating point unit in which floating point status information is encoded in the representations of the results generated thereby. The exemplary floating point unit includes a plurality of functional units, including an adder unit, a multiplier unit, a divider unit, a square root unit, a maximum/minimum unit, a comparator unit, a remainder unit, and a tester unit, all of which operate under control of functional unit control signals provided by a control unit. The present application is directed to an exemplary adder unit that can be used in floating point operations with the floating point unit described in related U.S. Patent Application Serial No. _____.

[045] FIG. 1 is a functional block diagram of an exemplary adder unit 10 constructed in accordance with an embodiment of the invention. Generally, the adder unit 10 receives two floating point operands and generates therefrom a result and, in some cases, floating point status information, with the floating point status information being encoded in and comprising part of the floating point representation of the result. Since the floating point status information comprises part of the floating point representation of the result, instead of being separate and apart from the result

as in prior art adder units, the implicit serialization that is required by maintaining the floating point status information separate and apart from the result can be obviated.

[046] The adder unit 10 encodes the floating point status information in results that are generated in certain formats. This will be illustrated in connection with FIG. 2. FIG. 2 depicts exemplary formats of floating point operands that the adder unit 10 may receive and of results that it generate. With reference to the embodiment illustrated in FIG. 2, seven formats are depicted, including a zero format 60, an underflow format 61, a denormalized format 62, a normalized non-zero format 63, an overflow format 64, an infinity format 65 and a not-a-number (NaN) format 66. The zero format 60 is used to represent the values "zero," or, more specifically, positive or negative zero, depending on the value of "s," the sign bit.

[047] The underflow format 61 provides a mechanism by which a functional unit 41 through 45 can indicate that the result of a computation is an underflow. In the underflow format, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary zero's, and the bits $f_{msb}...f_{lsb+1}$ of the fraction field, except for the least significant bit, are all binary zero's. The least significant bit f_{lsb} of the fraction field is a binary one.

[048] The denormalized format 62 and normalized non-zero format 63 are used to represent finite non-zero floating point values substantially along the lines of that described above in connection with IEEE Std. 754. In both formats 62 and 63, the sign bit "s" indicates whether the result is positive or negative. The bits $e_{msb}...e_{lsb}$ of the exponent field of the denormalized format 62 are all binary zero's, whereas the bits $e_{msb}...e_{lsb}$ of the exponent field of the normalized non-zero format 63 are mixed

one's and zero's, except that the exponent field of the normalized non-zero format 63 will not have a pattern in which bits $e_{msb}...e_{lsb+1}$ are all binary ones and the least significant bit e_{lsb} zero and the fraction field is all ones. In both formats 62 and 63, the bits $f_{msb}...f_{lsb}$ of the fraction field are not all binary zero's.

[049] The overflow format 64 provides a mechanism by which the adder unit 10 can indicate that the result of a computation is an overflow. In the overflow format 64, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb+1}$ of the exponent field are all binary ones, with the least significant bit e_{lsb} being zero. The bits $f_{msb}...f_{lsb}$ of the fraction field are all binary ones.

[050] The infinity format 65 provides a mechanism by which the adder unit 10 can indicate that the result is infinite. In the infinity format 65, the sign bit "s" indicates whether the result is positive or negative, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb}...f_{lsb+5}$ of the fraction field are all binary zero's. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags, which will be described below.

[051] The NaN format 66 provides a mechanism by which the adder unit 10 can indicate that the result is not a number. In the NaN format, the sign bit "s" can be any value, the bits $e_{msb}...e_{lsb}$ of the exponent field are all binary ones, and the bits $f_{msb}...f_{lsb+5}$ of the fraction field are not all binary zero's. The five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags which will be described below.

[052] As noted above, in values represented in both the infinity format 65 and the NaN format 66, the five low order bits $f_{lsb+4}...f_{lsb}$ of the fraction field are flags. In the formats used with the floating point unit 40 the five flags include the flags that

are defined by IEEE Std. 754, including an invalid operation flag "n," an overflow flag "o," an underflow flag "u," a division-by-zero flag "z," and an inexact flag "x." For example, a value in the NaN format 66 in which both the overflow flag "o" and the division-by-zero flag "z" are set, indicates that the value represents a result of a computation that an overflow (this from the overflow flag "o"), with the overflow caused by an attempt to divide by zero (this from the division-by-zero flag "z"). It should be noted that the flags provide the same status information as provided by a floating point status register (not shown) in a prior art floating point unit. Because the information is provided as part of the result, multiple instructions can be contemporaneously executed. In this manner, the floating point status information that may be generated during execution of one instruction, when stored, will not over-write previously-stored floating point status information generated during execution of another instruction.

[053] In addition to including status information in the five low-order bits $f_{lsb+4}...f_{lsb}$ of the fraction field for values in the NaN format 66, other information can also be encoded in the next five low-order bits $f_{lsb+9}...f_{lsb+5}$. If the value in the NaN format 66 is the result of an operation, the other information indicates the operation and types of operands that gave rise to the result. In one embodiment, the other information is associated with binary encoded values (BEV) of those bits $f_{lsb+9}...f_{lsb+5}$ as follows:

Bit Pattern of Result	BEV of $f_{lsb+4}...f_{lsb}$	Meaning
	0 or 1	no specific meaning
s11111111 10000000000000010nouzx	2	infinity minus infinity
s11111111 10000000000000011nouzx	3	OV minus OV
s11111111 10000000000000100nouzx	4	zero times infinity

s11111111 10000000000000101nouzx	5	UN times OV
	6 or 7	no specific meaning
s11111111 10000000000001000nouzx	8	zero divided by zero
s11111111 10000000000001001nouzx	9	infinity divided by infinity
s11111111 10000000000001010nouzx	10	UN divided by UN
s11111111 10000000000001011nouzx	11	OV divided by OV
s11111111 10000000000001100nouzx	12	square root of less than zero
	13-16	no specific meaning
s11111111 10000000000010001nouzx	17	remainder by zero
s11111111 10000000000010010nouzx	18	remainder by UN
s11111111 10000000000010011nouzx	19	remainder by OV
s11111111 10000000000010100nouzx	20	remainder infinity
s11111111 10000000000010101nouzx	21	remainder of infinity by zero
s11111111 10000000000010110nouzx	22	remainder of infinity by UN
s11111111 10000000000010111nouzx	23	remainder of infinity by OV
s11111111 10000000000011000nouzx	24	remainder of OV
s11111111 10000000000011001nouzx	25	remainder of OV by zero
s11111111 10000000000011010nouzx	26	remainder of OV by UN
s11111111 10000000000011011nouzx	27	remainder of OV by OV
	28-31	no specific meaning

[054] Where "OV" refers to an operand in the overflow format 64, "UN" refers to an operand in the underflow format 61 and "infinity" refers to an operand in the infinity format 65.

[055] Additionally, in the following, it will be assumed that the formats represent thirty-two bit values; extension to, for example, sixty-four bit values or values represented in other numbers of bits will be readily apparent to those skilled in the art.

[056] With this background, the structure and operation of the exemplary adder unit 10 will be described in connection with FIG. 1 and consistent with an embodiment of the invention. With reference to FIG. 1, the exemplary adder unit 10 includes two operand buffers 11A and 11B, respective operand analysis circuits 12A and 12B, an adder core 13, a result assembler 14 and an adder decision table logic circuit 15. The operand buffers 11A and 11B receive and store respective operands

from, for example, a set of registers (not shown) in a conventional manner. The adder core 13 receives the operands from the operand buffers 11A and 11B, except as described below, and rounding mode information from, for example, a rounding mode store 16. The adder core 13 then generates a result in accordance with IEEE Std. 754. Adder core 13 is conventional and will not be described in detail herein.

[057] Each operand analysis circuit 12A, 12B analyzes the operand in the respective buffer 11A, 11B and generates signals providing information relating thereto, which signals are provided to the adder decision table logic circuit 15. The result assembler 14 receives information from a number of sources, including the operand buffers 11A and 11B, adder core 13 and several predetermined value stores as described below. Under control of control signals from the adder decision table logic circuit 15, the result assembler 14 assembles the result, which is provided on a result bus 17. The result bus 17, in turn, may deliver the result to any convenient destination, such as a register in a register set (not shown), for storage or other use.

[058] The system for providing a floating point addition may comprise an analyzer circuit configured to determine a first status of a first floating point operand and a second status of a second floating point operand based upon data within the first floating point operand and the second floating point operand, respectively. The analyzer circuit may comprise buffer 11A, 11B and analysis circuit 12A, 12B. In addition, the system for providing a floating point addition may comprise a results circuit coupled to the analyzer circuit. The results circuit is configured to assert a resulting floating point operand containing the sum of the first floating point operand

and the second floating point operand and a resulting status embedded within the resulting floating point operand. The results circuit may comprise an adder circuit (comprising the adder core 13), the adder decision logic table circuit 15, and result assembler 14.

[059] Those skilled in the art will appreciate that the invention may be practiced in an electrical circuit comprising discrete electronic elements, packaged or integrated electronic chips containing logic gates, a circuit utilizing a microprocessor, or on a single chip containing electronic elements or microprocessors. It may also be provided using other technologies capable of performing logical operations such as, for example, AND, OR, and NOT, including but not limited to mechanical, optical, fluidic, and quantum technologies. In addition, the invention may be practiced within a general purpose computer or in any other circuits or systems as are known by those skilled in the art.

[060] As noted above, each operand analysis circuit 12A, 12B analyzes the operand in the respective buffer 11A, 11B and generates signals providing information relating thereto, which signals are provided to the adder decision table logic circuit 15. In the illustrated embodiment, each exemplary operand analysis circuit 12A, 12B, is implemented with a number of comparators, including:

[061] (i) a comparator 20A, 20B that generates an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in respective buffer 11 A, 11 B are all binary one's, which will be the case if the operand is in the infinity format 65 or the NaN format 66;

[062] (ii) a comparator 21A, 21B that generates an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in the respective buffer 11A, 11B are all binary one's, and the bit e_{lsb} is a binary zero, which may be the case if the operand is in the overflow format 64 or the normalized-non zero format 63;

[063] (iii) a comparator 22A, 22B that generates an asserted signal if the bits $e_{msb}...e_{lsb}$ of the exponent field of the operand in respective buffer 11A, 11B are all binary zero's, which will be the case if the operand is in the zero format 60, underflow format 61 or denormalized format 62;

[064] (iv) a comparator 23A, 23B that generates an asserted signal if the bits $f_{msb}....f_{lsb+5}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary one's, which may be the case if the operand is in the denormalized format 62, normalized non-zero format 63, overflow format 64, or NaN format 66;

[065] (v) a comparator 24A, 24B that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary zero's, which may be the case if the operand is in the zero format 60, underflow format 62, denormalized format 62, normalized non-zero format 63, or infinity format 65;

[066] (vi) a comparator 25A, 25B that generates an asserted signal if the bits $f_{lsb+4}....f_{lsb}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary ones, which may be the case if the operand is in the denormalized format 62, or normalized non-zero format 63, and which will be the case if in overflow format 64, or if all of the flags "n", "o", "u", "z", and "x" are set in the infinity format 65 or NaN format 66;

[067] (vii) a comparator 26A, 26B that generates an asserted signal if the bits $f_{lsb+4}....f_{lsb+1}$ of the fraction field of the operand in the respective buffer 11A, 11B are all binary zero's, and if the bit f_{lsb} of the fraction field is either a binary "zero" or "one", which will be the case if the operand is in the zero format 60 or the underflow format 61 and which may be the case if the operand is in the denormalized format 62 or normalized non-zero format 63 or if the flags "n," "o," "u," and "z" are clear and the flag "x" is either set or clear in the infinity format 65 or NaN format 66;

[068] (viii) a comparator 27A, 27B that generates an asserted signal if the bits $f_{lsb+4}...f_{lsb+1}$ of the fraction field of the operand in the respective buffer 11A, 11B are binary zero's and if the bit f_{lsb} of the fraction field is a binary "one", which will be the case if the operand is in the underflow format 61 and which may be the case if the operand is in the denormalized format 62 or normalized non-zero format 63 or if the flags "n," "o," "u," and "z" are clear and the flag "x" is set in the infinity format 65 or NaN format 66; and

[069] (ix) a comparator 28A, 28B that generates an asserted signal if all of the bits of the $f_{lsb+4}..f_{lsb}$ of the fraction field of the operand in the respective buffer 11A, 11B are binary zero's, which will be the case if the operand is in the zero format 60, may be the case if the operand is in the denormalized format 62 or normalized non-zero format 63, or if the flags "n" "o," "u," "z" and "x" are clear in the infinity format 65 or NaN format 66.

[070] In the illustrated embodiment, each operand analysis circuit also includes combinatorial logic elements that receive selected ones of the signals from

the comparators and generate asserted signals to provide indications as to certain characteristics of the respective operand, including:

[071] (x) an AND gate 30A, 30B that generates an asserted signal if the signals generated by both comparator 24A, 24B and comparator 26A, 26B are asserted, which will be the case if the respective operand is in the zero format 60 or underflow format 61, infinity format 65 or NaN format 66 if the flags "n", "o", "u", and "z" are clear and the flag "x" is either and set or clear and which may be the case if the operand is in the denormalized format 62 or normalized non-zero format 63;

[072] (xi) a NAND gate 31A, 31B that generates an asserted signal if the signal generated by comparator 22A, 22B is asserted and AND gate 30A, 30B is negated, which will be the case if the respective operand is in the denormalized format;

[073] (xii) a NAND gate 32A, 32B that generates an asserted signal if the signals generated by comparators 20A, 20B, 21A, 21B, and 22A, 22B are all negated which will be the case if the respective operand is in the normalized non-zero format 63;

[074] (xiii) an AND gate 37A, 37B that generates an asserted signal if the signals generated by both comparator 23A, 23B and comparator 25A, 25B are asserted, which will be the case if the respective operand is in the overflow format 64 and may be the case if the operand is in the denormalized format 62, normalized non-zero format 63 or NaN format 66;

[075] (xiv) a NAND gate 33A, 33B that generates an asserted signal if the signal generated by the comparator 21A ,21B is asserted and the signal generated

by the AND gate 37A, 37B is negated, thereby indicating that the exponent field of the respective operand indicates that the operand may be in either the normalized non-zero format 63 or the overflow format 64 and that, because the bits of the fraction portion are not all one's, the operand is in the normalized non-zero format 63 and not in the overflow format 64;

[076] (xv) an OR gate 34A, 34B that generates an asserted signal if the signal generated by any of NAND gate 31A, 31B; NAND gate 32A, 32B; or NAND gate 33A, 33B is generating an asserted signal, thereby indicating that the respective operand is in either the denormalized format 62 or the non-zero normalized format 63;

[077] (xvi) an AND gate 35A, 35B that generates an asserted signal if the signal generated by comparator 24A, 24B and comparator 28A, 28B are both asserted, thereby indicating that the fraction field of the respective operand is all zeros, which may be the case if the respective operand is in the zero format 60, normalized non-zero format 63, or infinity format 65;

[078] (xvii) an AND gate 36A, 36B that generates an asserted signal if the signal generated by comparator 24A, 24B and comparator 27A, 27B are both asserted, thereby indicating that the bits $f_{msb}...f_{lsb+1}$ of the fraction field of the respective operand are all binary zero's, and the bit f_{lsb} of the fraction field is a binary "one," which will be the case if the respective operand is in the underflow format 61, or infinity format 65, with "n", "o", "u", and "z" clear and the "x" flag set and may be the case if the operand is in the normalized format 63.

[079] In addition, the exemplary combinatorial logic includes a comparator 40 that generates an asserted signal if the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A represent a binary-encoded value that is larger than the binary-encoded value represented by bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B.

[080] Each operand analysis circuit 12A, 12B provides signals to the adder decision table logic 15 as follows:

(a) a signal representative of the sign bit "s" of the respective operand;

(b) the signal generated by the comparator 22A, 22B;

(c) the signal generated by the comparator 21A, 21B;

(d) the signal generated by the comparator 20A, 20B;

(e) the signal generated by the comparator 24A, 24B

(f) the signal generated by AND gate 35A; 35B;

(g) the signal generated by AND gate 36A, 36B;

(h) the signal generated by AND gate 37A, 37B; and

(i) the signal generated by the OR gate 34A, 34B.

In addition, the signal generated by comparator 40 is provided to the adder decision table logic 15, as are signals from rounding mode store 16 representative of the rounding mode. In addition, the adder core 13 generates an overflow signal, which is also provided to the adder decision table logic 15.

[081] The exemplary adder decision table logic 15 essentially generates control signals for controlling the result assembler 14 based upon the generated signals as described above. In addition, control signals generated by the adder

decision table logic 15 control sets of XOR gates 41A, 41B; 42A, 42B that control toggling of one or both of the two least significant bits f_{lsb}, f_{lsb+1} of the fraction field of the operands before they are provided to the adder core 13. As noted above, the result assembler 14 receives information from a number of sources, including the operand buffers 11A and 11B, adder core 13 and several predetermined value stores as described below. Under control of control signals from the adder decision table logic circuit 15, the result assembler 14 assembles the result, which is provided on a result bus 17. The result assembler 14 essentially assembles the result in four segments, including a sign segment that represents the sign bit of the result, an exponent segment that represents the exponent field of the result, a high-order fraction segment that represents the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result, and a low-order fraction segment that represents the five least significant bits $f_{lsb+4}...f_{lsb}$ of the result. It will be appreciated that the low-order fraction segment for results in the infinity format 65 and NaN format 66 correspond to the flags "n," "o," "u," "z" and "x".

[082] In the illustrated embodiment, the result assembler includes four elements, including a multiplexer 43, an exponent field selector 44, a high-order fraction field selector 45 and low-order fraction field combiner 46. The multiplexer 43 provides the sign segment of the result. The multiplexer 43 selectively couples one of a group of signals representative of the sign bit of the value generated by the adder core 13 and a signal from the adder decision table logic 15 under control of a control signal from the adder decision table logic 15. If the control signal is asserted, the multiplexer couples the signal representative of the sign bit of the value

generated by the adder core 13 to the result bus 17 as the sign of the result. As will be described below, if this control signal is asserted, the selectors 44, 45 and combiner 46 will also couple respective signals provided thereto by the adder core 13 to the result bus as respective segments of the results. Accordingly, if this control signal is asserted: 1) the selector 44 will couple the signals representative of the exponent field of the value generated by the adder core 13 to the result bus 17 as the exponent field of the result; 2) the selector 45 will couple the signals representative of the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the value generated by the adder core 13 to the result bus 17 as the corresponding bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result; and 3) combiner 46 will couple the signals representative of the bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the value generated by the adder core 13 to the result bus 17 as the corresponding bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result. Accordingly, if the adder decision table logic 15 generates this control signal, the result will correspond to the value generated by the adder core (except that some bits of the low-order fraction field may be forced to 1 by the combiner as described further below).

[083] In the illustrated embodiment, as noted above, the selector 44 couples exponent value signals representative of the exponent field of the result to the result bus 17. The selector 44 receives four sets of exponent field value signals, namely, the signals from the adder core 13 associated with the exponent field, as well as three sets of signals representative of three predetermined exponent field bit patterns as depicted in FIG. 1. It will be appreciated that these predetermined exponent field bit patterns correspond to the exponent fields associated with the

zero format 60, underflow format 61, overflow format 64, infinity format 65 and NaN format 66. In addition, the selector 44 receives four exponent field control signals from the adder decision table logic 15, each one of the control signals being associated with different ones of the sets of exponent field value signals. In enabling the result assembler 14 to assemble the result, the adder decision table logic will assert one of the four exponent field control signals, and the selector 44 will couple the set of exponent field value signals associated with the asserted exponent field control signal to the result bus 17 to provide the exponent field of the result.

[084] In the illustrated embodiment, the selector 45 couples high-order fraction field signals representative of the high-order fraction field bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result to the result bus 17. The selector 45 receives seven sets of high-order fraction field value signals, namely, the signals from the adder core 13 associated with the high-order fraction field, signals representative of bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in buffer 11B, as well as four sets of signals representative of four predetermined high-order fraction field bit patterns as depicted in FIG. 1. It will be appreciated that these predetermined high-order fraction field bit patterns correspond to the high-order fraction fields associated with the zero format 60, underflow format 61, overflow format 64, infinity format 65 and NaN format 66. In addition, the selector 45 receives seven high-order fraction field control signals from the adder decision table logic 15, each of the control signals being associated with different ones of the sets of high-order fraction field value signals. In enabling the result assembler 14 to assemble the result, the adder

decision table logic will assert one of the seven high-order fraction field control signals, and the selector 45 will couple the set of high-order fraction field value signals associated with the asserted high-order fraction field control signal to the result bus 17 to provide bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result.

[085] Similarly, the combiner 46 in the illustrated embodiment couples low-order fraction field value signals representative of the low-order fraction field bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result to the result bus 17. The combiner 46 receives four sets of low-order fraction field signals, namely, the signals from the adder core 13 associated with the low-order fraction field, signals representative of bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in buffer 11A, signals representative of bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the operand in buffer 11B, and one set of signals from the adder decision table logic 15. It will be appreciated that the set of signals provided by the adder decision table logic 15 will be used in controlling the condition of flags "n," "o," "u," "z," and "x" for those formats in which the low order bits $f_{lsb+4}...f_{lsb}$ represent flags. In addition, the sets of signals provided by the operands in buffers 11A and 11B may also represent the flags "n," "o," "u," "z," and "x". In addition, the combiner 46 receives three low-order fraction field control signals from the adder decision table logic 15, one control signal associated with the set of low-order fraction field value signals provided by the adder core and the two others associated with the sets of signals provided by the buffers 11A and 11B, respectively. In enabling the result assembler 14 to assemble the result, the adder decision table logic 15 may provide signals representative of the low-order fraction field and negate all of the low-order fraction field control signals, in which case the

signals representative of the low order fraction field provided by the adder decision table logic 15 will be coupled to the result bus 13 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result. Alternatively, the adder decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert one of the three low-order fraction field control signals. In this case the combiner 46 will couple the set of low-order fraction field value signals associated with the asserted low-order fraction field control signal to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result. As a further alternative, the adder decision table logic 15 may negate all of the low-order fraction field value signals provided thereby and assert more than one of the three low-order fraction field control signals. In this case the combiner 46 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result. As yet another alternative, the adder decision table logic 15 may assert one or more of the low-order fraction field value signals provided thereby and assert one or more of the three low-order fraction field control signals. In this case the combiner 46 will couple the bit-wise OR of the sets of low-order fraction field value signals associated with the asserted low-order fraction field control signals and the low-order fraction field value signals provided by the adder decision table logic 15 to the result bus 17 to provide bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result.

[086] In the illustrated embodiment, the combiner 46 comprises an OR gate 50 and three AND gates 51 through 53. (Each gate in the diagram actually represents five such gates, one for each bit position $f_{lsb+4}...f_{lsb}$ but for sake of clarity

and to avoid confusion, the diagram illustrates them as a single gate.) The AND gates 51-53 receive the low-order fraction field value signals from the adder core 13 and operand buffers 11A and 11B respectively, as well as the respective low-order fraction field control signal. These and gates 51-53 perform a bit-wise AND operation to, if the respective low-order fraction field control signal is asserted, couple the low-order fraction field value signals to a respective input of OR gate 50. The OR gate 50, whose output is connected to the result bus 17, performs a bit-wise OR operation in connection with the signals that it receives from the AND gates 51-53 and the low-order fraction field value signals provided by the adder decision table logic 15. If the adder decision table logic 15 negates all of the low-order fraction field control signals, the AND gates 51-53 will block the low-order fraction field value signals that they receive and the signals provided by the OR gate 50 will conform to the low-order fraction field value signals provided by the adder decision table logic 15.

[087] On the other hand, if the adder decision table logic 15 asserts one or more of the low-order fraction field control signals, the AND gates 51-53 that receive the asserted low-order fraction field control signal will couple the low-order fraction field value signals that they receive to the OR gate 50 and the other AND gates will block the low-order fraction field signal that they receive. As will be described below, under some circumstances, the adder decision table logic 15 will assert two low-order fraction field control signals to enable two sets of low-order fraction field value signals to be coupled to the OR gate 50. In that case, the OR gate will perform a bit-wise OR operation in connection with signals representing respective bits of the

low-order fraction field. This adder decision table logic 15 will assert two low-order fraction signals if, for example, both operands in operand buffers 11A and 11B are in NaN format to enable the respective flags "n", "o", "u", "z", and "x" to be ORed together.

[088] However, if the low-order fraction field value signals provided by the adder decision table logic 15 are negated, the low-order fraction field value signals provided by the OR gate 50 will conform to the low-order fraction field signals provided by the AND gate that receives the asserted low-order fraction field control signal.

[089] As noted above, the exemplary adder decision table logic 15 generates control signals for controlling the multiplexer 43 and various selectors 44-45 and combiner 46 that make up the result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffers 11A and 11B before they are presented to the adder core 13. The control signals generated by the adder decision table logic 15 are such as to enable the result to be assembled in the desired format 60-66 having status information embedded within the result itself. Before proceeding further, it would be helpful to describe the results that are to be generated by the adder unit 10.

[090] Generally, exemplary results generated by the adder unit 10 are described in the table depicted in FIG. 3 and are consistent with an embodiment of the present invention. The table of FIG. 3 (using the key set forth below) describes the results to be generated by the adder unit 10. In that table, one skilled in the art will appreciate that "+P" or "+Q" means any finite positive nonzero representable

value other than +UN (that is, a value in the underflow format 61, with the sign bit "s" being "zero") and +OV (that is, a value in the overflow format 64 with the sign bit "s" being "zero"). "-P" or "-Q" means any finite negative nonzero representable value other than -UN (that is, a value in the underflow format 61, with the sign bit "s" being "one") and -OV (that is, a value in the overflow format 64, with the sign bit being "one"). Additionally, "NaN" means any value whose exponent field is 11111111, other than one of the values represented by +∞ (that is, a value in the infinity format 65, with the sign bit "s" being "zero") and - ∞ (that is, a value in the infinity format 65, with the sign bit "s" being "one").

[091] Key to symbols in the table with exemplary results depicted in FIG. 3 are as follows:

[092] (a) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the two operands.

[093] (b) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand with bit pattern 01001 (to indicate overflow and inexact).

[094] (c) The result is -∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand. Those skilled in the art will appreciate that even if the other operand is -UN or +UN, it is intentional that the low five bits of the -∞ operand not be ORed with 00101 to indicate underflow and inexact.

[095] (d) For "round toward positive infinity," the result is +∞. For "round toward negative infinity," the result is -∞. In either of these two cases, the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the two operands. For all other rounding modes, the result is the positive NaN value 0 11111111 10000000000000000101ouzx (to indicate "infinity minus infinity" with the invalid operation flag set). The four least significant bits $f_{lsb+3}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the four least significant bits $f_{lsb+3}...f_{lsb}$ of the fraction fields of the two operands.

[096] (e) The result is a copy of the NaN operand, except that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the two operand.

[097] (f) For "round toward plus infinity," the result is the same as if -OV were replaced by -HUGE and +UN were replaced by +TINY. In other words, the result in the illustrated embodiment will be 1 11111110 11111111111111111111101). For all other rounding modes, the result is -OV.

[098] (g) For "round toward plus infinity," the result is the same as if -OV were replaced by -HUGE. For all other rounding modes, the result is -OV.

[099] (h) For "round toward plus infinity," the result is +OV. For "round toward minus infinity," the result is -OV. For all other rounding modes, the result is the positive NaN value, which in the illustrated embodiment is 0 11111111 10000000000000001111001 (to indicate "OV minus OV" with the invalid operation "n," overflow "o," and inexact "x" flags set).

[0100] (i) The result is $+\infty$, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand with 01001 (to indicate overflow and inexact).

[0101] (j) The result is a copy of the NaN operand, except that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the NaN operand with 01001 (to indicate overflow and inexact).

[0102] (k) As computed in accordance with IEEE Std. 754. However, the result is -OV if (1) overflow occurs or (2) if the rounding mode is "round toward minus infinity" and the mathematical sum is less than -HUGE.

[0103] (l) For "round toward plus infinity," the result is the same as if -UN were replaced by -0. For "round toward minus infinity," the result is the same as if -UN were replaced by -TINY; for all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0104] (m) For "round toward plus infinity," the result is the same as if +UN were replaced by +TINY. For "round toward minus infinity," the result is the same as if +UN were replaced by +0. For all other rounding modes, the result is as computed in accordance with IEEE Std. 754.

[0105] (n) As computed in accordance with IEEE Std. 754. If IEEE Std 754 would compute the result as 1 00000000 00000000000000000000001, then so does this embodiment of the invention (but this embodiment calls it -UN and considers it to be underflow). If IEEE 754 would compute the result as 0 00000000

00000000000000000000001 then so does this embodiment (but this embodiment calls it +UN and considers it to be underflow.

[0106] (o) For "round toward minus infinity," the result is the same as if +OV were replaced by +HUGE. For all other rounding modes, the result is +OV.

[0107] (p) The result is +∞ with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being equal to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the infinite operand. Those skilled in the art will appreciate that even if the other operand is -UN or +UN, it is intentional that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the +∞ operand not be ORed with 00101 to indicate underflow and inexact.

[0108] (q) The result is a copy of the NaN operand.

[0109] (r) For "round toward minus infinity," the result is the same as if each -UN were replaced by -TINY (that is, the result will be -2*TINY). For all other rounding modes, the result is -UN.

[0110] (s) For "round toward minus infinity," the result is -UN. For all other rounding modes, the result is +UN.

[0111] (t) For "round toward minus infinity," the result is the same as if +OV were replaced by +HUGE and -UN were replaced by -TINY. That is, the result will be 0 11111110 11111111111111111111101). For all other rounding modes, the result is +OV.

[0112] (u) The result is a copy of the NaN operand, except that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of

the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field NaN operand with 00101 (to indicate underflow and inexact).

[0113] (v) For "round toward minus infinity," the result is -0. For all other rounding modes, the result is +0. This is as in accordance with IEEE Std. 754.

[0114] (w) For "round toward plus infinity," the result is the same as if each +UN were replaced by +TINY (that is, the result will be +2*TINY). For all other rounding modes, the result is +UN.

[0115] (x) As computed in accordance with IEEE Std.754, except that the result is +OV if (1) overflow occurs or, (2) if the rounding mode is "round toward plus infinity" and the mathematical sum is greater than +HUGE.

[0116] (y) The result is +∞, with the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result being the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operands.

[0117] (z) The result is a copy of the NaN operand that has the larger value in its fraction field, except that the five least significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result are the bitwise OR of the five least significant bits $f_{lsb+4}...f_{lsb}$ of the operands and the sign bit "s" of the result is "one" if and only if the sign bits of the two NaN operands are both "ones."

[0118] Those skilled in the art will appreciate that, with adder unit 10 operating according to the table depicted in FIG. 3, addition is commutative. This is true even with those cases where one or both operands are values in the NaN format 66.

[0119] As noted above, adder decision table logic 15 generates control signals for controlling the multiplexer 43 and various selectors 44-45 and combiner

46 making up the exemplary result assembler 14 and for controlling the toggling of the signals representing the low-order bits f_{lsb+1} and f_{lsb} from the operand buffers 11A and 11B before they are presented to the adder core 13. The particular signals that the adder decision table logic 15 will generate depend on the signals provided thereto by the operand buffers 11A and 11B representing the states of the respective sign bits, the operand analysis circuits 12A and 12B, comparator 40, rounding mode store 16, and the overflow signal from the adder core 13. The series of input signals received by the adder decision table logic 15 are as follows:

[0120] (a) a signal from the operand buffer 11A that is asserted if the sign of the operand therein is negative;

[0121] (b) a signal from the comparator 22A that is asserted if the exponent field of the operand in operand buffer 11A has an exponent field that has the bit pattern 00000000;

[0122] (c) a signal from the comparator 21A that is asserted if the exponent field of the operand in operand buffer 11A has an exponent field that has the bit pattern 11111110;

[0123] (d) a signal from the comparator 20A that is asserted if the exponent field of the operand in operand buffer 11A has an exponent field that has the bit pattern 11111111;

[0124] (e) a signal from the comparator 24A that is asserted if the operand in operand buffer 11A has a high-order fraction field that has the bit pattern 00000000000000000;

[0125] (f) a signal from the AND gate 35A that is asserted if the operand in operand buffer 11A has both high- and low-order fraction fields with all 0-bits;

[0126] (g) a signal from the AND gate 36A that is asserted if the operand in operand buffer 11A has high- and low-order fraction fields with the collective bit pattern 00000000000000000000001;

[0127] (h) a signal from the AND gate 37A that is asserted if the operand buffer 11A has high- and low-order fraction fields with the collective bit pattern 11111111111111111111111;

[0128] (i) a signal from the OR gate 34A that is asserted if any of the following signals asserted:

[0129] (1) a signal from NAND gate 31A that is asserted if the exponent field of the operand in operand buffer 11A has the bit pattern 00000000 (which will be the case if the signal from comparator 22A is asserted) and the high- and low-order fraction fields of the operand in operand buffer 11A collectively have a bit pattern in which at least one bit, other than the least significant bit, is "1" (which will be the case if the signal from AND gate 30A is negated);

[0130] (2) a signal from NAND gate 32A is asserted, which will be the case if the exponent field of the operand in operand buffer 11A does not have the bit pattern 00000000 or 11111110 or 11111111; and

[0131] (3) a signal from NAND gate 33A is asserted, which will be the case if the exponent field of the operand in operand buffer 11A has bit pattern 11111110 and the high- and low-order fraction fields are collectively not all 1-bits;

[0132] (j) a signal from the operand buffer 11B that is asserted if the sign of the operand therein is negative;

[0133] (k) a signal from the comparator 22B that is asserted if the exponent field of the operand in operand buffer 11B has an exponent field that has the bit pattern 00000000;

[0134] (l) a signal from the comparator 21B that is asserted if the exponent field of the operand in operand buffer 11B has an exponent field that has the bit pattern 11111110;

[0135] (m) a signal from the comparator 20B that is asserted if the exponent field of the operand in operand buffer 11B has an exponent field that has the bit pattern 11111111;

[0136] (n) a signal from the comparator 24B that is asserted if the operand in operand buffer 11B has a high-order fraction field that has the bit pattern 00000000000000000;

[0137] (o) a signal from the AND gate 35B that is asserted if the operand in operand buffer 11B has both high- and low-order fraction fields with all 0-bits;

[0138] (p) a signal from the AND gate 36B that is asserted if the operand in operand buffer 11B has high- and low-order fraction fields with the collective bit pattern 00000000000000000000001;

[0139] (q) a signal from the AND gate 37B that is asserted if the operand in operand buffer 11B has high- and low-order fraction fields with the collective bit pattern 11111111111111111111111;

[0140] (r) a signal from the OR gate 34B that is asserted if any of the following signals are asserted:

[0141] (1) a signal from NAND gate 31B that is asserted if the exponent field of the operand in operand buffer 11B has the bit pattern 00000000 (which will be the case if the signal from comparator 22B is asserted) and the high- and low-order fraction fields of the operand in operand buffer 11B collectively have a bit pattern in which at least one bit, other than the least significant bit, is "1" (which will be the case if the signal from AND gate 30B is negated);

[0142] (2) a signal from NAND gate 32B is asserted, which will be the case if the exponent field of the operand in operand buffer 11B does not have the bit pattern 00000000 or 11111110 or 11111111; and

[0143] (3) a signal from NAND gate 33B is asserted, which will be the case if the exponent field of the operand in operand buffer 11B has bit pattern 11111110 and the high- and low-order fraction fields are collectively not all 1-bits;

[0144] (s) a signal from comparator 40 that is asserted if the binary-encoded value of the bits comprising the high-order fraction field of the operand in operand buffer 11A is greater than the binary-encoded value of the bits comprising the high-order fraction field of the operand in operand buffer 11B;

[0145] (t) a signal from the rounding mode store that is asserted if the rounding mode is either "round toward plus infinity" or "round toward minus infinity";

[0146] (u) a signal from the rounding mode store that is asserted if the rounding mode is either "round toward zero" or "round toward minus infinity"; and

[0147] (v) an "overflow" signal from the adder core 13.

[0148] In response to these signals, the exemplary adder decision logic table 15 generates the following;

[0149] (1) a signal that, if asserted, enables the second least-significant bit of the operand in operand buffer 11A to be toggled before presented to the adder core 13;

[0150] (2) a signal that, if asserted, enables the least significant bit of the operand in operand buffer 11A to be toggled before presented to the adder core 13;

[0151] (3) a signal that, if asserted, enables the second-least significant bit of the operand in operand buffer 11B to be toggled before presented to the adder core 13;

[0152] (4) a signal that, if asserted, enables the least significant bit of the operand in operand buffer 11B to be toggled before presented to the adder core 13;

[0153] (5) a signal that, if asserted, enables the sign bit, the exponent field, and the high part of the fraction of the result to be provided by the adder core 13; moreover, the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the output provided by the adder core 13 will contribute to the five least significant bits $f_{lsb+4}...f_{lsb}$ of the result;

[0154] (6) a sign literal signal; if signal (5) is negated, then the sign bit of the result will be represented by this signal;

[0155] (7) a signal that, if asserted, will enable the exponent field of the result to have the bit pattern 00000000;

[0156] (8) a signal that, if asserted, the will enable the exponent field of the result to have the bit pattern 11111110;

[0157] (9) a signal that, if asserted, will enable the exponent field of the result to have the bit pattern 11111111;

[0158] (10) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the high-order portion of the fraction of the operation in operand buffer 11A;

[0159] (11) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the high-order portion of the fraction of the operation in operand buffer 11B;

[0160] (12) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 0000000000000000;

[0161] (13) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 1111111111111111;

[0162] (14) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 1000000000000010 (to represent a NaN value "infinity minus infinity");

[0163] (15) a signal that, if asserted, will enable the high-order fraction of the result to correspond to the bit pattern 1000000000000011 (to represent a NaN value "overflow minus overflow");

[0164] (16) a signal that, if asserted, will enable the low-order fraction field of the operand in output buffer 11A to contribute to the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result;

[0165] (17) a signal that, if asserted, will enable the low-order fraction field of the operand in output buffer 11B to contribute to the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result; and

[0166] (18)-(22) signals that always contribute to the five least-significant bits $f_{lsb+4}...f_{lsb}$ of the fraction field of the result.

[0167] The specific patterns of output signals (1) through (22) generated by the exemplary adder decision table logic 15 in response to patterns of input signals (a) through (v) are depicted in FIGS. 4A through 4E. Generally, in those FIGS., each row represents conditions of the output signals (1) through (22) that are generated by the adder decision table logic 15 in response to one pattern of input signals (a) through (v). In each row, the indicia to the left of the asterisk (*) represent the pattern of input signals (a) through (v) and the indicia to the right of the asterisk represent the pattern of output signals (1) through (22) with a "1" indicating that the respective input or output signal is asserted, a "0" indicating that the respective input or output signal is negated and a "-" indicating that the respective input signal may be either negated or asserted. Each row is further annotated with an indication as to the respective format 60 through 66 of the operand in the respective operand buffers 11A and 11B and the format of the result.

[0168] Referring now to Fig. 4A, a discussion of the first row of input signal values and output signal values follows:

[0169] (A) for the three patterns to the left of the asterisk:

[0170] (i) the first pattern "1--10----" indicates that signals (a) and (d) are asserted, signal (e) is negated, and the other signals (b), (c) and (f) through (i) may

be either asserted or negated with the pattern indicating a value in the NaN format 66 with a negative sign ("[-NaN]");

[0171] (ii) the second pattern "1--10----" indicates that signals (j) and (m) are asserted, signal (n) is negated, and the other signals (k), (l) and (o) through (r) may be either asserted or negated, with the pattern indicating a value in the NaN format 66 with a negative sign ("[-NaN]"); and

[0172] (iii) the third pattern "1 -- -" indicates that the binary-encoded value of the high-order bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11A is greater than the binary-encoded value of the high-order bits $f_{msb}...f_{lsb+5}$ of the fraction field of the operand in operand buffer 11B; and

[0173] (B) for the seven patterns to the right of the asterisk:

[0174] (i) the pattern "0000" to the immediate right of the asterisk indicates that the signals provided to XOR gates 41A, 42A, 41B and 42B, which control the toggling of the low-order bits f_{lsb+1} and f_{lsb} of the fraction fields of the operands in operand buffers 11A and 11B before being presented to adder core 13, are all negated;

[0175] (ii) the next "0" indicates that the signal provided to multiplexer 43, selectors 44 and 45 and AND gate 51 is negated to ensure that the output from adder core 13 will not contribute the result;

[0176] (iii) the next "1" corresponds to the signal provided by the adder decision logic table 15 to one input of multiplexer 43 and indicates that the signal is asserted, to ensure that the sign bit of the result will be asserted;

[0177] (iv) the next pattern "001" indicates that the signal will be asserted that will enable selector 44 to couple signals representative of the pattern 11111111 to the result bus 17 and that the signals associated with the other patterns 00000000 and 1111110 will be negated;

[0178] (v) the next pattern "100000" indicates that the signal will be asserted that will enable selector 45 to couple the signals associated with the bits $f_{msb}...f_{lsb+5}$ (comprising the high-order fraction field of the operand in buffer 11A) to the result bus 17 as the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result;

[0179] (vi) the next pattern "11" indicates that the signals provided by the adder decision table logic 15 will provide asserted signals to both AND gates 52 and 53, enabling both AND gates to couple signals received thereby from both operand buffers 11A and 11B to the OR gate 50; and

[0180] (vii) the last pattern "00000" indicates that the signals provided by the adder decision table logic 15 to the OR gate 50 are all negated; in that case, the OR gate will perform a bit-wise OR operation in connection with those signals and the signals provided thereto by AND gates 51 and 53; the negated signal described in (B)(ii) provides that the signals provided by AND gate 51 are also negated, in which case the signals coupled by OR gate 50 to result bus 17 will correspond to the OR of the bits $f_{lsb+4}...f_{lsb}$ from the fraction field of the operands in operand buffers 11A and 11B.

[0181] Additionally, the legend "[-NaN op1 f1|f2]" indicates that the result value is in the NaN format 66, with a negative value, with the bits $f_{msb}...f_{lsb+5}$ of the fraction field of the result corresponding to bits $f_{msb}...f_{lsb+5}$ of the fraction field of the

operand in operand buffer 11A (op1) and the bits $f_{lsb+4}...f_{lsb}$ of the result corresponding to the OR of the bits $f_{lsb+4}...f_{lsb}$ of the fraction fields of the operands (representing flags) in both operand buffers 11A and 11 B (f1|f2). It should be noted that this corresponds to the result represented by symbol (z) in the table depicted on FIG. 3.

[0182] In the context of the above discussion, the other rows of Fig, 4A and all rows in Figs. 4B , 4C, 4D and 4E will be apparent to those skilled in the art.

[0183] Adder decision table logic 15 may be implemented by many different circuit elements that will be apparent to those skilled in the art, including, but not limited to programmable logic arrays.

[0184] One of ordinary skill in the art will recognize that other formats and bit patterns could be used to represent the floating point operand formats without departing from the principles of the present invention. One of ordinary skill in the art will also recognize that the floating point status information contained in the operands could easily be represented by other bit combinations (not shown) without departing from the principles of the present invention. For example, more or fewer bits could be used, a subset or superset of the exemplary status bits could be used, or the most significant bits of an operand (or some other subset of bits) could be used to indicate the floating point status information, instead of the least significant bits illustrated.

[0185] It will be appreciated that a system in accordance with an embodiment of the invention can be constructed in whole or in part from special purpose hardware or a general purpose computer system, or any combination thereof, any

portion of which may be controlled by a suitable program. Any program may in whole or in part comprise part of or be stored on the system in a conventional manner, or it may in whole or in part be provided in to the system over a network or other mechanism for transferring information in a conventional manner. In addition, it will be appreciated that the system may be operated and/or otherwise controlled by means of information provided by an operator using operator input elements (not shown) which may be connected directly in the system in which may transfer the information to the system over a network or other mechanism for transferring information in a conventional manner.

[0186] The foregoing description has been limited to a specific embodiment of this invention. It will be apparent, however, that various variations and modifications may be made to the invention, with the attainment of some or all of the advantages of the invention. It is the object of the appended claims to cover these and such other variations and modifications as come within the true spirit and scope of the invention.

[0187] Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.